```
// This file is part of Visual D
//
// Visual D integrates the D programming language into Visual Studio
// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved
//
// Distributed under the Boost Software License, Version 1.0.
// See accompanying file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt

module visuald.propertypage;

import visuald.windows;

import sdk.win32.objbase;
import sdk.win32.winreg;
import sdk.vsi.vsshell;
import sdk.vsi.vsshell80;

import visuald.comutil;
import visuald.logutil;
import visuald.dpackage;
import visuald.dproject;
import visuald.dllmain;
import visuald.config;
import visuald.winctrl;
import visuald.hierarchy;
import visuald.hierutil;
import visuald.pkgutil;
import visuald.chiernode;
import visuald.register;

import stdext.array;
import stdext.path;
import std.array;
import std.string;
import std.conv;
import std.algorithm;
```

```
// version = DParserOption;

class PropertyWindow : Window
{
    this(Widget parent, uint style, string title, PropertyPage page)
    {
        mPropertyPage = page;
        super(parent, style, title);
    }

    override int WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)
    {
        import sdk.win32.commctrl;

        switch (uMsg) {
            case WM_SIZE:
                mPropertyPage.updateSizes();
                break;

            case TCN_SELCHANGING:
            case TCN_SELCHANGE:
                // Return FALSE to allow the selection to change.
                auto tc = cast(TabControl) this;
                return FALSE;

            default:
                break;
        }
        return super.WindowProc(hWnd, uMsg, wParam, lParam);
    }

    PropertyPage mPropertyPage;
}

abstract class PropertyPage : DisposingComObject, IPropertyPage, IVsPropertyPage,
IVsPropertyPage2
{
    /*const*/ int kPageWidth = 370;
```

```
/*const*/ int kPageHeight = 210;
/*const*/ int kMargin = 2;
/*const*/ int kLabelWidth = 120;
/*const*/ int kTextHeight = 20;
/*const*/ int kLineHeight = 23;
/*const*/ int kLineSpacing = 2;
/*const*/ int kNeededLines = 11;

override HRESULT QueryInterface(in IID* riid, void** pvObject)
{
    if(queryInterface!(IPropertyPage) (this, riid, pvObject))
        return S_OK;
    if(queryInterface!(IVsPropertyPage) (this, riid, pvObject))
        return S_OK;
    if(queryInterface!(IVsPropertyPage2) (this, riid, pvObject))
        return S_OK;
    return super.QueryInterface(riid, pvObject);
}

override void Dispose()
{
    mResizableWidgets = mResizableWidgets.init;

    mSite = release(mSite);

    foreach(obj; mObjects)
        release(obj);
    mObjects.length = 0;

    mDlgFont = deleteDialogFont(mDlgFont);
}

override int SetPageSite(
    /* [in] */ IPropertyPageSite pPageSite)
{
    mixin(LogCallMix);
    mSite = release(mSite);
    mSite = addref(pPageSite);
    return S_OK;
```

```
    }

    override int Activate(
        /* [in] */ in HWND hWndParent,
        /* [in] */ in RECT *pRect,
        /* [in] */ in BOOL bModal)
    {
        mixin(LogCallMix);

        if(mWindow)
            return returnError(E_FAIL);
        return _Activate(new Window(hWndParent), pRect, bModal != 0);
    }

    int _Activate(Window win, const(RECT) *pRect, bool bModal)
    {
        updateEnvironmentFont();
        if(!mDlgFont)
            mDlgFont = newDialogFont();

        mWindow = win;
        mCanvas = new Window(mWindow);
        DWORD color = GetSysColor(COLOR_BTNFACE);
        mCanvas.setBackground(color);

        // create with desired size to get proper alignment, then resize to parent later
        mCanvas.setRect(kMargin, kMargin, kPageWidth - 2 * kMargin, kPageHeight - 2 *
kMargin);

        // avoid closing canvas (but not dialog) if pressing esc in MultiLineEdit controls
        //mCanvas.cancelCloseDelegate ~= delegate bool(Widget c) { return true; };

        mCanvas.commandDelegate = &OnCommand;

        CreateControls();
        UpdateControls();
        updateSizes();

        mEnableUpdateDirty = true;
```

```
    return S_OK;
}

extern(D) void OnCommand(Widget w, int cmd)
{
    UpdateDirty(true);
}

override int Deactivate()
{
    mixin(LogCallMix);
    if(mWindow)
    {
        auto win = mWindow;
        mCanvas = null;
        mWindow = null;
        win.Dispose();
    }

    return S_OK;
    //return returnError(E_NOTIMPL);
}

void updateSizes()
{
    if (!mWindow || !mCanvas)
        return;

    RECT r, pr;
    mCanvas.GetWindowRect(&r);
    mWindow.GetWindowRect(&pr);
    int pageWidth = pr.right - pr.left - 2 * kMargin;
    int pageHeight = pr.bottom - pr.top - 2 * kMargin;

    if (r.right - r.left == pageWidth && r.bottom - r.top == pageHeight)
        return;

    mCanvas.setRect(kMargin, kMargin, pageWidth, pageHeight);
```

```
    updateResizableWidgets(mCanvas);
}

void updateResizableWidgets(Widget w)
{
    if (auto patt = w in mResizableWidgets)
        patt.resizeWidget(w);

    foreach(c; w.children)
        updateResizableWidgets(c);

}

void addResizableWidget(Widget w, Attachment att)
{
    AttachData attData = AttachData(att);
    attData.initFromWidget(w);
    mResizableWidgets[w] = attData;
}

void refreshResizableWidget(Widget w)
{
    if (auto att = w in mResizableWidgets)
        att.initFromWidget(w);
}

void addTextPath(Text ctrl, string path, string sep)
{
    string imp = ctrl.getText();
    if(!imp.empty() && !imp.endsWith(sep))
        imp ~= sep;
    imp ~= quoteFilename(path);
    ctrl.setText(imp);
}

void addBrowsePath(Text ctrl, bool dir, string reldir, string sep, string title, string filter = null)
{
    string path;
    if(dir)
```

```
        path = browseDirectory(mCanvas.hwnd, title, reldir);
    else
        path = browseFile(mCanvas.hwnd, title, filter, reldir);
    if (!path.empty)
    {
        if(reldir)
            path = makeRelative(path, reldir);

        addTextPath(ctrl, path, sep);
    }
}

void calcMetric()
{
    updateEnvironmentFont();

    if(!mDlgFont)
        mDlgFont = newDialogFont();
    HWND hwnd = GetDesktopWindow();
    HDC dc = GetDC(hwnd);
    SelectObject(dc, mDlgFont);
    TEXTMETRIC tm;
    GetTextMetrics(dc, &tm);
    ReleaseDC(hwnd, dc);

    int fHeight = tm.tmHeight;
    int fWidth = tm.tmAveCharWidth;

    kPageWidth = fWidth * 75 + 2 * kMargin;
    kLabelWidth = fWidth * 22;
    mUnindentCheckBox = kLabelWidth;

    kLineSpacing = 2;
    kTextHeight = fHeight + 4;
    kLineHeight = kTextHeight + kLineSpacing + 1;
    kPageHeight = kLineHeight * kNeededLines + 2 * kMargin;
}

override int GetPageInfo(
```

```
    /* [out] */ PROPPAGEINFO *pPageInfo)
{
    mixin(LogCallMix);

    if(pPageInfo.cb < PROPPAGEINFO.sizeof)
        return E_INVALIDARG;

    calcMetric();
    pPageInfo.cb = PROPPAGEINFO.sizeof;
    pPageInfo.pszTitle = string2OLESTR("Title");
    pPageInfo.size = visuald.comutil.SIZE(kPageWidth, kPageHeight);
    pPageInfo.pszHelpFile = string2OLESTR("HelpFile");
    pPageInfo.pszDocString = string2OLESTR("DocString");
    pPageInfo.dwHelpContext = 0;

    return S_OK;
}

override int SetObjects(
    /* [in] */ in ULONG cObjects,
    /* [size_is][in] */ IUnknown *ppUnk)
{
    mixin(LogCallMix2);

    foreach(obj; mObjects)
        release(obj);
    mObjects.length = 0;
    for(uint i = 0; i < cObjects; i++)
        mObjects ~= addref(ppUnk[i]);

    if(mWindow)
    {
        mEnableUpdateDirty = false;
        UpdateControls();
        mEnableUpdateDirty = true;
    }

    return S_OK;
}
```

```
override int Show(
    /* [in] */ in UINT nCmdShow)
{
    logCall("%s.Show(nCmdShow=%s)", this, _toLog(nCmdShow));
    if(mWindow)
        mWindow.setVisible(true);
    return S_OK;
    //return returnError(E_NOTIMPL);
}

override int Move(
    /* [in] */ in RECT *pRect)
{
    mixin(LogCallMix);
    updateSizes();
    return S_OK; //returnError(E_NOTIMPL);
}

override int Help(
    /* [in] */ in wchar* pszHelpDir)
{
    logCall("%s.Help(pszHelpDir=%s)", this, _toLog(pszHelpDir));
    return returnError(E_NOTIMPL);
}

override int TranslateAccelerator(
    /* [in] */ in MSG *pMsg)
{
    mixin(LogCallMix2);
    if(mSite)
        return mSite.TranslateAccelerator(pMsg);
    return returnError(E_NOTIMPL);
}

// IVsPropertyPage
override int get_CategoryTitle(
    /* [in] */ in UINT iLevel,
    /* [retval][out] */ BSTR *pbstrCategory)
```

```
    {
        logCall("%s.get_CategoryTitle(iLevel=%s, pbstrCategory=%s)", this, _toLog(iLevel),
_toLog(pbstrCategory));
        switch(iLevel)
        {
        case 0:
            if(GetCategoryName().length == 0)
                return S_FALSE;
            *pbstrCategory = allocBSTR(GetCategoryName());
            break;
        case 1:
            return S_FALSE;
            //*pbstrCategory = allocBSTR("CategoryTitle1");
        default:
            break;
        }
        return S_OK;
    }

    // IVsPropertyPage2
    override int GetProperty(
        /* [in] */ in VSPPPID propid,
        /* [out] */ VARIANT *pvar)
    {
        mixin(LogCallMix);
        switch(propid)
        {
        case VSPPPID_PAGENAME:
            pvar.vt = VT_BSTR;
            pvar.bstrVal = allocBSTR(GetPageName());
            return S_OK;
        default:
            break;
        }
        return returnError(DISP_E_MEMBERNOTFOUND);
    }

    override int SetProperty(
        /* [in] */ in VSPPPID propid,
```

```
        /* [in] */ in VARIANT var)
    {
        mixin(LogCallMix);
        return returnError(E_NOTIMPL);
    }

    ////////////////////////////////////////
    void UpdateDirty(bool bDirty)
    {
        if(mEnableUpdateDirty && mSite)
            mSite.OnStatusChange(PROPPAGESTATUS_DIRTY |
PROPPAGESTATUS_VALIDATE);
    }

    static int getWidgetWidth(Widget w, int def)
    {
        RECT pr;
        if(w && w.GetWindowRect(&pr))
            return pr.right - pr.left;
        return def;
    }

    void AddControl(string label, Widget w)
    {
        AddControl(label, w, null, 0);
    }

    void AddControl(string label, Widget w, Button btn)
    {
        AddControl(label, w, btn, 0);
    }

    void AddControl(string label, Widget w, short attachY)
    {
        AddControl(label, w, null, attachY);
    }

    void AddControl(string label, Widget w, Button btn, short resizeY)
    {
```

```
int x = kLabelWidth;
auto cb = cast(CheckBox) w;
auto tc = cast(TabControl) w;
auto mt = cast(MultiLineText) w;
//if(cb)
//    cb.cmd = 1; // enable actionDelegate

int lines = 1;
if(mt || tc)
    lines = mLinesPerMultiLine;

int pageWidth = getWidgetWidth(w ? w.parent : null, kPageWidth);
if (btn)
    pageWidth -= kLineHeight;
int labelWidth = 0;
int margin = tc ? 0 : kMargin;
if(label.length)
{
    Label lab = new Label(w ? w.parent : null, label);
    int off = ((kLineHeight - kLineSpacing) - 16) / 2;
    labelWidth = w ? kLabelWidth : pageWidth - 2*margin;
    lab.setRect(0, mLineY + off, labelWidth, kLineHeight - kLineSpacing);

    if(mAttachY > 0)
    {
        Attachment att = kAttachNone;
        att.vdiv = 1000;
        att.top = att.bottom = mAttachY;
        addResizableWidget(lab, att);
    }
}
else if (cb || tc)
{
    x -= mUnindentCheckBox;
}

int h = lines * kLineHeight - kLineSpacing;
if(cast(Text) w && lines == 1)
{
```

```
        h = kTextHeight;
    }
    else if(cb)
        h -= 2;
    else if(tc)
        h += tc.getFrameHeight() - kLineHeight;
    //else if(cast(ComboBox) w)
    //    h -= 4;

    int yspacing = (lines * kLineHeight - kLineSpacing - h) / 2;
    int y = mLineY + max(0, yspacing);
    if(w)
    {
        w.setRect(x, y, pageWidth - 2*margin - labelWidth, h);
        Attachment att = kAttachLeftRight;
        att.vdiv = 1000;
        att.top = mAttachY;
        att.bottom = cast(short)(mAttachY + resizeY);
        addResizableWidget(w, att);
    }
    if(btn)
    {
        btn.setRect(pageWidth - kMargin, y, kLineHeight, kLineHeight - kLineSpacing);
        Attachment att = kAttachRight;
        att.vdiv = 1000;
        att.top = att.bottom = mAttachY;
        addResizableWidget(btn, att);
    }
    mLineY += max(h, lines * kLineHeight);
    mAttachY += resizeY;
}

void AddLabel(string lab)
{
    auto w = new Label(mCanvas, lab);
    int pageWidth = getWidgetWidth(mCanvas, kPageWidth);
    int off = ((kLineHeight - kLineSpacing) - 16) / 2;
    w.setRect(0, mLineY + off, pageWidth - 2*kMargin, kLineHeight - kLineSpacing);
    Attachment att = kAttachLeftRight;
```

```
    att.vdiv = 1000;
    att.top = att.bottom = mAttachY;
    addResizableWidget(w, att);
    mLineY += kLineHeight;
}

void AddHorizontalLine()
{
    auto w = new Label(mCanvas);
    w.AddWindowStyle(SS_ETCHEDFRAME, SS_TYPEMASK);
    w.setRect(0, mLineY + 2, getWidgetWidth(mCanvas, kPageWidth) - 2*kMargin, 2);
    Attachment att = kAttachLeftRight;
    att.vdiv = 1000;
    att.top = att.bottom = mAttachY;
    addResizableWidget(w, att);
    mLineY += 6;
}

int changeOption(V)(V val, ref V optval, ref V refval)
{
    if(refval == val)
        return 0;
    optval = val;
    return 1;
}
int changeOptionDg(V)(V val, void delegate (V optval) setdg, V refval)
{
    if(refval == val)
        return 0;
    setdg(val);
    return 1;
}

abstract void CreateControls();
abstract void UpdateControls();
abstract string GetCategoryName();
abstract string GetPageName();

AttachData[Widget] mResizableWidgets;
```

```
    HFONT mDlgFont;
    IUnknown[] mObjects;
    IPropertyPageSite mSite;
    Window mWindow;
    Window mCanvas;
    bool mEnableUpdateDirty;
    int mLineY;
    short mAttachY = 0; // fraction of 1000
    int mLinesPerMultiLine = 4;
    int mUnindentCheckBox = 120; //16;
}

////////////////////////////////////////////////////////////////////////////
class ProjectPropertyPage : PropertyPage, ConfigModifiedListener
{
    abstract void SetControls(ProjectOptions options);
    abstract int  DoApply(ProjectOptions options, ProjectOptions refoptions);

    override HRESULT QueryInterface(in IID* riid, void** pvObject)
    {
        //if(queryInterface!(ConfigModifiedListener) (this, riid, pvObject))
        //    return S_OK;
        return super.QueryInterface(riid, pvObject);
    }

    override void UpdateControls()
    {
        if(ProjectOptions options = GetProjectOptions())
            SetControls(options);
    }

    override void Dispose()
    {
        if(auto cfg = GetConfig())
            cfg.RemoveModifiedListener(this);

        super.Dispose();
    }
```

```
override void OnConfigModified()
{
}

override int SetObjects(/* [in] */ in ULONG cObjects,
                        /* [size_is][in] */ IUnknown *ppUnk)
{
    if(auto cfg = GetConfig())
        cfg.RemoveModifiedListener(this);

    int rc = super.SetObjects(cObjects, ppUnk);

    if(auto cfg = GetConfig())
        cfg.AddModifiedListener(this);

    return rc;
}

Config GetConfig()
{
    if(mObjects.length > 0)
    {
        auto config = ComPtr!(Config)(mObjects[0]);
        return config;
    }
    return null;
}
ProjectOptions GetProjectOptions()
{
    if(auto cfg = GetConfig())
        return cfg.GetProjectOptions();
    return null;
}

string GetProjectDir()
{
    if(auto cfg = GetConfig())
        return cfg.GetProjectDir();
    return null;
```

```
    }

    /*override*/ int IsPageDirty()
    {
        mixin(LogCallMix);
        if(mWindow)
            if(ProjectOptions options = GetProjectOptions())
            {
                scope ProjectOptions opt = new ProjectOptions(false, false);
                return DoApply(opt, options) > 0 ? S_OK : S_FALSE;
            }
        return S_FALSE;
    }

    /*override*/ int Apply()
    {
        mixin(LogCallMix);

        if(ProjectOptions refoptions = GetProjectOptions())
        {
            // make a copy, otherwise changes will no longer be detected after the first
configuration
            auto refopt = clone(refoptions);
            for(int i = 0; i < mObjects.length; i++)
            {
                auto config = ComPtr!(Config)(mObjects[i]);
                if(config)
                {
                    DoApply(config.ptr.GetProjectOptions(), refopt);
                    config.SetDirty();
                }
            }
            return S_OK;
        }
        return returnError(E_FAIL);
    }
}

class NodePropertyPage : PropertyPage
```

```
{
    abstract void SetControls(CFileNode node);
    abstract int  DoApply(CFileNode node, CFileNode refnode);

    override void UpdateControls()
    {
        if(CFileNode node = GetNode())
            SetControls(node);
    }

    CFileNode GetNode()
    {
        if(mObjects.length > 0)
        {
            auto node = ComPtr!(CFileNode)(mObjects[0]);
            if(node)
                return node;
        }
        return null;
    }

    /*override*/ int IsPageDirty()
    {
        mixin(LogCallMix);
        if(mWindow)
            if(CFileNode node = GetNode())
            {
                scope CFileNode n = newCom!CFileNode("");
                return DoApply(n, node) > 0 ? S_OK : S_FALSE;
            }
        return S_FALSE;
    }

    /*override*/ int Apply()
    {
        mixin(LogCallMix);

        if(CFileNode refnode = GetNode())
        {
```

```
            for(int i = 0; i < mObjects.length; i++)
            {
                auto node = ComPtr!(CFileNode)(mObjects[i]);
                if(node)
                {
                    DoApply(node, refnode);
                    if(CProjectNode pn = cast(CProjectNode) node.GetRootNode())
                        pn.SetProjectFileDirty(true);
                }
            }
            return S_OK;
        }
        return returnError(E_FAIL);
    }
}

class ResizablePropertyPage : PropertyPage
{
    void SetWindowSize(int x, int y, int w, int h)
    {
        mixin(LogCallMix);
        if(mCanvas)
            mCanvas.setRect(x, y, w, h);
    }
}

class GlobalPropertyPage : ResizablePropertyPage
{
    abstract void SetControls(GlobalOptions options);
    abstract int  DoApply(GlobalOptions options, GlobalOptions refoptions);

    this(GlobalOptions options)
    {
        mOptions = options;
    }

    override void UpdateControls()
    {
        if(GlobalOptions options = GetGlobalOptions())
```

```
        SetControls(options);
    }

    GlobalOptions GetGlobalOptions()
    {
        return mOptions;
    }

    /*override*/ int IsPageDirty()
    {
        mixin(LogCallMix);
        if(mWindow)
            if(GlobalOptions options = GetGlobalOptions())
            {
                scope GlobalOptions opt = new GlobalOptions;
                return DoApply(opt, options) > 0 ? S_OK : S_FALSE;
            }
        return S_FALSE;
    }

    /*override*/ int Apply()
    {
        mixin(LogCallMix);

        if(GlobalOptions options = GetGlobalOptions())
        {
            DoApply(options, options);
            options.saveToRegistry();
            return S_OK;
        }
        return returnError(E_FAIL);
    }

    GlobalOptions mOptions;
}

////////////////////////////////////////////////////////////////////////////
class CommonPropertyPage : ProjectPropertyPage
{
```

```
    override string GetCategoryName() { return ""; }
    override string GetPageName() { return "General"; }

    override void CreateControls()
    {
        AddControl("Build System",  mCbBuildSystem = new ComboBox(mCanvas, [ "Visual D",
"dsss", "rebuild" ], false));
        mCbBuildSystem.setSelection(0);
        mCbBuildSystem.setEnabled(false);
    }
    override void SetControls(ProjectOptions options)
    {
    }
    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        return 0;
    }

    ComboBox mCbBuildSystem;
}

class GeneralPropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return ""; }
    override string GetPageName() { return "General"; }

    __gshared const float[] selectableVersions = [ 1, 2 ];

    override void CreateControls()
    {
        string[] versions;
        foreach(ver; selectableVersions)
            versions ~= "D" ~ to!(string)(ver);
        //versions[$-1] ~= "+";

        AddControl("Compiler",      mCompiler = new ComboBox(mCanvas, [ "DMD", "GDC",
"LDC" ], false));
        AddControl("D-Version",     mDVersion = new ComboBox(mCanvas, versions, false));
        AddControl("Output Type",   mCbOutputType = new ComboBox(mCanvas,
```

```
                                                [ "Executable", "Library", "DLL" ], false));
        AddControl("Subsystem",     mCbSubsystem = new ComboBox(mCanvas,
                                                [ "Not set", "Console", "Windows", "Native", "Posix" ],
false));
        AddControl("Output Path",   mOutputPath = new Text(mCanvas));
        AddControl("Intermediate Path", mIntermediatePath = new Text(mCanvas));
        AddControl("Files to clean", mFilesToClean = new Text(mCanvas));
        AddControl("Compilation",   mSingleFileComp = new ComboBox(mCanvas,
            [ "Combined compile and link", "Single file compilation",
              "Separate compile and link", "Compile only (use Post-build command to link)" ],
false));
    }

    override void SetControls(ProjectOptions options)
    {
        int ver = 0;
        while(ver < selectableVersions.length - 1 && selectableVersions[ver+1] <=
options.Dversion)
            ver++;
        mDVersion.setSelection(ver);

        mCompiler.setSelection(options.compiler);
        mSingleFileComp.setSelection(options.compilationModel);
        mCbOutputType.setSelection(options.lib);
        mCbSubsystem.setSelection(options.subsystem);
        mOutputPath.setText(options.outdir);
        mIntermediatePath.setText(options.objdir);
        mFilesToClean.setText(options.filesToClean);
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        float ver = selectableVersions[mDVersion.getSelection()];
        int changes = 0;
        changes += changeOption(cast(uint) mSingleFileComp.getSelection(),
options.compilationModel, refoptions.compilationModel);
        changes += changeOption(cast(ubyte) mCbOutputType.getSelection(), options.lib,
refoptions.lib);
        changes += changeOption(cast(ubyte) mCbSubsystem.getSelection(),
```

```
options.subsystem, refoptions.subsystem);
        changes += changeOption(cast(ubyte) mCompiler.getSelection(), options.compiler,
refoptions.compiler);
        changes += changeOption(ver, options.Dversion, refoptions.Dversion);
        changes += changeOption(mOutputPath.getText(), options.outdir, refoptions.outdir);
        changes += changeOption(mIntermediatePath.getText(), options.objdir, refoptions.objdir);
        changes += changeOption(mFilesToClean.getText(), options.filesToClean,
refoptions.filesToClean);
        return changes;
    }

    ComboBox mCompiler;
    ComboBox mSingleFileComp;
    ComboBox mCbOutputType;
    ComboBox mCbSubsystem;
    ComboBox mDVersion;
    Text mOutputPath;
    Text mIntermediatePath;
    Text mFilesToClean;
}

class DebuggingPropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return ""; }
    override string GetPageName() { return "Debugging"; }

    enum ID_DBGCOMMAND = 1020;
    enum ID_DBGDIR = 1021;

    extern(D) override void OnCommand(Widget w, int cmd)
    {
        switch(cmd)
        {
            case ID_DBGCOMMAND:
                if(auto file = browseFile(mCanvas.hwnd, "Select executable",
"Executables\0*.exe\0All Files\0*.*\0"))
                    mCommand.setText(file);
                break;
            case ID_DBGDIR:
```

```
            if(auto dir = browseDirectory(mCanvas.hwnd, "Select working directory"))
                mWorkingDir.setText(dir);
            break;
        default:
            break;
    }
    super.OnCommand(w, cmd);
}

override void CreateControls()
{
    Label lbl;
    auto btn = new Button(mCanvas, "...", ID_DBGCOMMAND);
    AddControl("Command",           mCommand = new Text(mCanvas), btn);
    AddControl("Command Arguments", mArguments = new Text(mCanvas));
    btn = new Button(mCanvas, "...", ID_DBGDIR);
    AddControl("Working Directory", mWorkingDir = new Text(mCanvas), btn);
    AddControl("",                  mAttach = new CheckBox(mCanvas, "Attach to running process"));
    AddControl("Remote Machine",    mRemote = new Text(mCanvas));
    AddControl("Debugger",          mDebugEngine = new ComboBox(mCanvas, [ "Visual Studio", "Mago", "Visual Studio (x86 Mixed Mode)" ], false));
    AddControl("",                  mStdOutToOutputWindow = new CheckBox(mCanvas, "Redirect stdout to output window"));
    AddControl("Run without debugging", lbl = new Label(mCanvas, ""));
    AddControl("",                  mPauseAfterRunning = new CheckBox(mCanvas, "Pause when program finishes"));

    lbl.AddWindowExStyle(WS_EX_STATICEDGE);
    lbl.AddWindowStyle(SS_ETCHEDFRAME, SS_TYPEMASK);
    int left, top, w, h;
    if(lbl.getRect(left, top, w, h))
        lbl.setRect(left, top + h / 2 - 1, w, 2);
    refreshResizableWidget(lbl);
}

override void UpdateDirty(bool bDirty)
{
    super.UpdateDirty(bDirty);
```

```
        EnableControls();
    }

    void EnableControls()
    {
        mStdOutToOutputWindow.setEnabled(mDebugEngine.getSelection() != 1);
    }

    override void SetControls(ProjectOptions options)
    {
        mCommand.setText(options.debugtarget);
        mArguments.setText(options.debugarguments);
        mWorkingDir.setText(options.debugworkingdir);
        mAttach.setChecked(options.debugattach);
        mRemote.setText(options.debugremote);
        mDebugEngine.setSelection(options.debugEngine);
        mStdOutToOutputWindow.setChecked(options.debugStdOutToOutputWindow);
        mPauseAfterRunning.setChecked(options.pauseAfterRunning);

        EnableControls();
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
        changes += changeOption(mCommand.getText(), options.debugtarget,
refoptions.debugtarget);
        changes += changeOption(mArguments.getText(), options.debugarguments,
refoptions.debugarguments);
        changes += changeOption(mWorkingDir.getText(), options.debugworkingdir,
refoptions.debugworkingdir);
        changes += changeOption(mAttach.isChecked(), options.debugattach,
options.debugattach);
        changes += changeOption(mRemote.getText(), options.debugremote,
refoptions.debugremote);
        changes += changeOption(cast(ubyte)mDebugEngine.getSelection(),
options.debugEngine, refoptions.debugEngine);
        changes += changeOption(mStdOutToOutputWindow.isChecked(),
options.debugStdOutToOutputWindow, options.debugStdOutToOutputWindow);
```

```
        changes += changeOption(mPauseAfterRunning.isChecked(),
options.pauseAfterRunning, options.pauseAfterRunning);
        return changes;
    }

    Text mCommand;
    Text mArguments;
    Text mWorkingDir;
    Text mRemote;
    CheckBox mAttach;
    ComboBox mDebugEngine;
    CheckBox mStdOutToOutputWindow;
    CheckBox mPauseAfterRunning;
}

class DmdGeneralPropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return "Compiler"; }
    override string GetPageName() { return "General"; }

    enum ID_IMPORTPATH = 1030;
    enum ID_STRINGIMPORTPATH = 1031;

    void addImportDir(Text ctrl, string title)
    {
        addBrowsePath(ctrl, true, GetProjectDir(), ";", title);
    }

    extern(D) override void OnCommand(Widget w, int cmd)
    {
        switch(cmd)
        {
            case ID_IMPORTPATH:
                addImportDir(mAddImports, "Add import path");
                break;
            case ID_STRINGIMPORTPATH:
                addImportDir(mStringImports, "Add string import path");
                break;
            default:
```

```
            break;
        }
        super.OnCommand(w, cmd);
    }


    override void CreateControls()
    {
        AddControl("",                  mDepImports = new CheckBox(mCanvas, "Add import paths
of project dependencies"));
        auto btn = new Button(mCanvas, "+", ID_IMPORTPATH);
        AddControl("Additional Import Paths", mAddImports = new Text(mCanvas), btn);
        btn = new Button(mCanvas, "+", ID_STRINGIMPORTPATH);
        AddControl("String Import Paths", mStringImports = new Text(mCanvas), btn);
        AddControl("Version Identifiers", mVersionIdentifiers = new Text(mCanvas));
        AddControl("Debug Identifiers",   mDebugIdentifiers = new Text(mCanvas));
        AddHorizontalLine();
        AddControl("",                  mOtherDMD = new CheckBox(mCanvas, "Use other
compiler"));
        AddControl("Compiler Path",       mCompilerPath = new Text(mCanvas));
        AddHorizontalLine();
        AddControl("C/C++ Compiler Cmd",  mCCCmd = new Text(mCanvas));
        AddControl("",                  mTransOpt = new CheckBox(mCanvas, "Translate D options
(debug, optimizations)"));
    }

    override void UpdateDirty(bool bDirty)
    {
        super.UpdateDirty(bDirty);

        EnableControls();
    }

    void EnableControls()
    {
        mCompilerPath.setEnabled(mOtherDMD.isChecked());
    }

    override void SetControls(ProjectOptions options)
```

```
    {
        mDepImports.setChecked(options.addDepImp);

        mAddImports.setText(options.imppath);
        mStringImports.setText(options.fileImppath);
        mVersionIdentifiers.setText(options.versionids);
        mDebugIdentifiers.setText(options.debugids);

        mOtherDMD.setChecked(options.otherDMD);
        mCompilerPath.setText(options.program);
        mCCCmd.setText(options.cccmd);
        mTransOpt.setChecked(options.ccTransOpt);

        EnableControls();
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
        changes += changeOption(mDepImports.isChecked(), options.addDepImp,
refoptions.addDepImp);
        changes += changeOption(mAddImports.getText(), options.imppath, refoptions.imppath);
        changes += changeOption(mStringImports.getText(), options.fileImppath,
refoptions.fileImppath);
        changes += changeOption(mVersionIdentifiers.getText(), options.versionids,
refoptions.versionids);
        changes += changeOption(mDebugIdentifiers.getText(), options.debugids,
refoptions.debugids);

        changes += changeOption(mOtherDMD.isChecked(), options.otherDMD,
refoptions.otherDMD);
        changes += changeOption(mCompilerPath.getText(), options.program,
refoptions.program);
        changes += changeOption(mCCCmd.getText(), options.cccmd, refoptions.cccmd);
        changes += changeOption(mTransOpt.isChecked(), options.ccTransOpt,
refoptions.ccTransOpt);
        return changes;
    }
```

```
    CheckBox mDepImports;
    Text mAddImports;
    Text mStringImports;
    Text mVersionIdentifiers;
    Text mDebugIdentifiers;

    CheckBox mOtherDMD;
    Text mCompilerPath;
    Text mCCCmd;
    CheckBox mTransOpt;
}

class DmdDebugPropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return "Compiler"; }
    override string GetPageName() { return "Debug"; }

    enum ID_BROWSECV2PDB = 1010;

    extern(D) override void OnCommand(Widget w, int cmd)
    {
        switch(cmd)
        {
            case ID_BROWSECV2PDB:
                if(auto file = browseFile(mCanvas.hwnd, "Select cv2pdb executable",
"Executables\0*.exe\0All Files\0*.*\0"))
                    mPathCv2pdb.setText(file);
                break;
            default:
                break;
        }
        super.OnCommand(w, cmd);
    }

    override void CreateControls()
    {
        string[] dbgInfoOpt = [ "None", "Symbolic (suitable for Mago)", "Symbolic (suitable for VS
debug engine)", "Symbolic (suitable for selected debug engine)" ];
        AddControl("Debug Mode", mDebugMode = new ComboBox(mCanvas, [ "On (enable
```

```
debug statements, asserts, invariants and constraints)",
                                                "Off (disable asserts, invariants and constraints)",
                                                "Default (enable asserts, invariants and
constraints)" ], false));
        AddControl("Debug Info", mDebugInfo = new ComboBox(mCanvas, dbgInfoOpt, false));
        AddHorizontalLine();
        string[] cv2pdbOpt = [ "Never", "If recommended for selected debug engine", "Always" ];
        AddControl("Run cv2pdb", mRunCv2pdb = new ComboBox(mCanvas, cv2pdbOpt,
false));
        auto btn = new Button(mCanvas, "...", ID_BROWSECV2PDB);
        AddControl("Path to cv2pdb", mPathCv2pdb = new Text(mCanvas), btn);
        AddControl("",          mCv2pdbPre2043  = new CheckBox(mCanvas, "Assume old
associative array implementation (before dmd 2.043)"));
        AddControl("",          mCv2pdbNoDemangle = new CheckBox(mCanvas, "Do not
demangle symbols"));
        AddControl("",          mCv2pdbEnumType = new CheckBox(mCanvas, "Use enumerator
types"));
        AddControl("More options", mCv2pdbOptions  = new Text(mCanvas));
    }

    override void UpdateDirty(bool bDirty)
    {
        super.UpdateDirty(bDirty);
        EnableControls();
    }

    void EnableControls()
    {
        mRunCv2pdb.setEnabled(mCanRunCv2PDB);
        bool runcv2pdb = mCanRunCv2PDB && mRunCv2pdb.getSelection() > 0;

        mPathCv2pdb.setEnabled(runcv2pdb);
        mCv2pdbOptions.setEnabled(runcv2pdb);
        mCv2pdbEnumType.setEnabled(runcv2pdb);
        mCv2pdbPre2043.setEnabled(runcv2pdb);
        mCv2pdbNoDemangle.setEnabled(runcv2pdb);
    }

    override void SetControls(ProjectOptions options)
```

```
    {
        mDebugMode.setSelection(options.release);
        mDebugInfo.setSelection(options.symdebug);
        mRunCv2pdb.setSelection(options.runCv2pdb);
        mPathCv2pdb.setText(options.pathCv2pdb);
        mCv2pdbOptions.setText(options.cv2pdbOptions);
        mCv2pdbPre2043.setChecked(options.cv2pdbPre2043);
        mCv2pdbNoDemangle.setChecked(options.cv2pdbNoDemangle);
        mCv2pdbEnumType.setChecked(options.cv2pdbEnumType);

        mCanRunCv2PDB = options.compiler != Compiler.DMD || (!options.isX86_64 &&
!options.mscoff);
        EnableControls();
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
        changes += changeOption(cast(ubyte) mDebugMode.getSelection(), options.release,
refoptions.release);
        changes += changeOption(cast(ubyte) mDebugInfo.getSelection(), options.symdebug,
refoptions.symdebug);
        changes += changeOption(cast(ubyte) mRunCv2pdb.getSelection(), options.runCv2pdb,
refoptions.runCv2pdb);
        changes += changeOption(mPathCv2pdb.getText(), options.pathCv2pdb,
refoptions.pathCv2pdb);
        changes += changeOption(mCv2pdbOptions.getText(), options.cv2pdbOptions,
refoptions.cv2pdbOptions);
        changes += changeOption(mCv2pdbPre2043.isChecked(), options.cv2pdbPre2043,
refoptions.cv2pdbPre2043);
        changes += changeOption(mCv2pdbNoDemangle.isChecked(),
options.cv2pdbNoDemangle, refoptions.cv2pdbNoDemangle);
        changes += changeOption(mCv2pdbEnumType.isChecked(), options.cv2pdbEnumType,
refoptions.cv2pdbEnumType);
        return changes;
    }

    bool mCanRunCv2PDB;
    ComboBox mDebugMode;
```

```
    ComboBox mDebugInfo;
    ComboBox mRunCv2pdb;
    Text mPathCv2pdb;
    CheckBox mCv2pdbPre2043;
    CheckBox mCv2pdbNoDemangle;
    CheckBox mCv2pdbEnumType;
    Text mCv2pdbOptions;
}

class DmdCodeGenPropertyPage : ProjectPropertyPage
{
    this()
    {
        kNeededLines = 12;
    }

    override string GetCategoryName() { return "Compiler"; }
    override string GetPageName() { return "Code Generation"; }

    override void CreateControls()
    {
        mUnindentCheckBox = kLabelWidth;
        AddControl("", mProfiling     = new CheckBox(mCanvas, "Insert Profiling Hooks"));
        AddControl("", mCodeCov       = new CheckBox(mCanvas, "Generate Code Coverage"));
        AddControl("", mUnitTests     = new CheckBox(mCanvas, "Generate Unittest Code"));
        AddHorizontalLine();
        AddControl("", mOptimizer     = new CheckBox(mCanvas, "Run Optimizer"));
        AddControl("", mNoboundscheck = new CheckBox(mCanvas, "No Array Bounds
Checking"));
        AddControl("", mInline        = new CheckBox(mCanvas, "Expand Inline Functions"));
        AddHorizontalLine();
        AddControl("", mNoFloat       = new CheckBox(mCanvas, "No Floating Point Support"));
        AddControl("", mGenStackFrame = new CheckBox(mCanvas, "Always generate stack
frame (DMD 2.056+)"));
        AddControl("", mStackStomp    = new CheckBox(mCanvas, "Add stack stomp code
(DMD 2.062+)"));
        AddControl("", mAllInst       = new CheckBox(mCanvas, "Generate code for all template
instantiations (DMD 2.064+)"));
    }
```

```
    override void SetControls(ProjectOptions options)
    {
        mProfiling.setChecked(options.trace);
        mCodeCov.setChecked(options.cov);
        mOptimizer.setChecked(options.optimize);
        mNoboundscheck.setChecked(options.noboundscheck);
        mUnitTests.setChecked(options.useUnitTests);
        mInline.setChecked(options.useInline);
        mNoFloat.setChecked(options.nofloat);
        mGenStackFrame.setChecked(options.genStackFrame);
        mStackStomp.setChecked(options.stackStomp);
        mAllInst.setChecked(options.allinst);

        mNoboundscheck.setEnabled(options.Dversion > 1);
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
        changes += changeOption(mCodeCov.isChecked(), options.cov, refoptions.cov);
        changes += changeOption(mProfiling.isChecked(), options.trace, refoptions.trace);
        changes += changeOption(mOptimizer.isChecked(), options.optimize,
refoptions.optimize);
        changes += changeOption(mNoboundscheck.isChecked(), options.noboundscheck,
refoptions.noboundscheck);
        changes += changeOption(mUnitTests.isChecked(), options.useUnitTests,
refoptions.useUnitTests);
        changes += changeOption(mInline.isChecked(), options.useInline, refoptions.useInline);
        changes += changeOption(mNoFloat.isChecked(), options.nofloat, refoptions.nofloat);
        changes += changeOption(mGenStackFrame.isChecked(), options.genStackFrame,
refoptions.genStackFrame);
        changes += changeOption(mStackStomp.isChecked(), options.stackStomp,
refoptions.stackStomp);
        changes += changeOption(mAllInst.isChecked(), options.allinst, refoptions.allinst);
        return changes;
    }

    CheckBox mCodeCov;
```

```
    CheckBox mProfiling;
    CheckBox mOptimizer;
    CheckBox mNoboundscheck;
    CheckBox mUnitTests;
    CheckBox mInline;
    CheckBox mNoFloat;
    CheckBox mGenStackFrame;
    CheckBox mStackStomp;
    CheckBox mAllInst;
}

class DmdMessagesPropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return "Compiler"; }
    override string GetPageName() { return "Messages"; }

    override void CreateControls()
    {
        mUnindentCheckBox = kLabelWidth;
        AddControl("", mWarnings      = new CheckBox(mCanvas, "Enable Warnings"));
        AddControl("", mInfoWarnings  = new CheckBox(mCanvas, "Enable Informational
Warnings (DMD 2.041+)"));
        AddHorizontalLine();
        AddControl("", mUseDeprecated = new CheckBox(mCanvas, "Silently Allow Deprecated
Features"));
        AddControl("", mErrDeprecated = new CheckBox(mCanvas, "Use of Deprecated
Features causes Error (DMD 2.061+)"));
        AddHorizontalLine();
        AddControl("", mVerbose       = new CheckBox(mCanvas, "Verbose Compile"));
        AddControl("", mVtls          = new CheckBox(mCanvas, "Show TLS Variables"));
        AddControl("", mVgc           = new CheckBox(mCanvas, "List all gc allocations including
hidden ones (DMD 2.066+)"));
        AddControl("", mIgnorePragmas = new CheckBox(mCanvas, "Ignore Unsupported
Pragmas"));
        AddControl("", mCheckProperty = new CheckBox(mCanvas, "Enforce Property Syntax
(DMD 2.055+)"));
    }

    override void SetControls(ProjectOptions options)
```

```
    {
        mWarnings.setChecked(options.warnings);
        mInfoWarnings.setChecked(options.infowarnings);
        mVerbose.setChecked(options.verbose);
        mVtls.setChecked(options.vtls);
        mVgc.setChecked(options.vgc);
        mUseDeprecated.setChecked(options.useDeprecated);
        mErrDeprecated.setChecked(options.errDeprecated);
        mIgnorePragmas.setChecked(options.ignoreUnsupportedPragmas);
        mCheckProperty.setChecked(options.checkProperty);

        mVtls.setEnabled(options.Dversion > 1);
        mVgc.setEnabled(options.Dversion > 1);
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
        changes += changeOption(mWarnings.isChecked(), options.warnings,
refoptions.warnings);
        changes += changeOption(mInfoWarnings.isChecked(), options.infowarnings,
refoptions.infowarnings);
        changes += changeOption(mVerbose.isChecked(), options.verbose, refoptions.verbose);
        changes += changeOption(mVtls.isChecked(), options.vtls, refoptions.vtls);
        changes += changeOption(mVgc.isChecked(), options.vgc, refoptions.vgc);
        changes += changeOption(mUseDeprecated.isChecked(), options.useDeprecated,
refoptions.useDeprecated);
        changes += changeOption(mErrDeprecated.isChecked(), options.errDeprecated,
refoptions.errDeprecated);
        changes += changeOption(mIgnorePragmas.isChecked(),
options.ignoreUnsupportedPragmas, refoptions.ignoreUnsupportedPragmas);
        changes += changeOption(mCheckProperty.isChecked(), options.checkProperty,
refoptions.checkProperty);
        return changes;
    }

    CheckBox mWarnings;
    CheckBox mInfoWarnings;
    CheckBox mVerbose;
```

```
    CheckBox mVtls;
    CheckBox mVgc;
    CheckBox mUseDeprecated;
    CheckBox mErrDeprecated;
    CheckBox mIgnorePragmas;
    CheckBox mCheckProperty;
}

class DmdDocPropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return "Compiler"; }
    override string GetPageName() { return "Documentation"; }

    override void CreateControls()
    {
        AddControl("", mGenDoc = new CheckBox(mCanvas, "Generate documentation"));
        AddControl("Documentation file", mDocFile = new Text(mCanvas));
        AddControl("Documentation dir", mDocDir = new Text(mCanvas));
        AddControl("CanDyDOC module", mModulesDDoc = new Text(mCanvas));

        AddControl("", mGenHdr = new CheckBox(mCanvas, "Generate interface headers"));
        AddControl("Header file",  mHdrFile = new Text(mCanvas));
        AddControl("Header directory",  mHdrDir = new Text(mCanvas));

        AddControl("", mGenJSON = new CheckBox(mCanvas, "Generate JSON file"));
        AddControl("JSON file",  mJSONFile = new Text(mCanvas));
    }

    override void UpdateDirty(bool bDirty)
    {
        super.UpdateDirty(bDirty);
        EnableControls();
    }

    void EnableControls()
    {
        mDocDir.setEnabled(mGenDoc.isChecked());
        mDocFile.setEnabled(mGenDoc.isChecked());
        mModulesDDoc.setEnabled(mGenDoc.isChecked());
```

```
        mHdrDir.setEnabled(mGenHdr.isChecked());
        mHdrFile.setEnabled(mGenHdr.isChecked());

        mJSONFile.setEnabled(mGenJSON.isChecked());
    }

    override void SetControls(ProjectOptions options)
    {
        mGenDoc.setChecked(options.doDocComments);
        mDocDir.setText(options.docdir);
        mDocFile.setText(options.docname);
        mModulesDDoc.setText(options.modules_ddoc);
        mGenHdr.setChecked(options.doHdrGeneration);
        mHdrDir.setText(options.hdrdir);
        mHdrFile.setText(options.hdrname);
        mGenJSON.setChecked(options.doXGeneration);
        mJSONFile.setText(options.xfilename);

        EnableControls();
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
        changes += changeOption(mGenDoc.isChecked(), options.doDocComments,
refoptions.doDocComments);
        changes += changeOption(mDocDir.getText(), options.docdir, refoptions.docdir);
        changes += changeOption(mDocFile.getText(), options.docname, refoptions.docname);
        changes += changeOption(mModulesDDoc.getText(), options.modules_ddoc,
refoptions.modules_ddoc);
        changes += changeOption(mGenHdr.isChecked(), options.doHdrGeneration,
refoptions.doHdrGeneration);
        changes += changeOption(mHdrDir.getText(), options.hdrdir, refoptions.hdrdir);
        changes += changeOption(mHdrFile.getText(), options.hdrname, refoptions.hdrname);
        changes += changeOption(mGenJSON.isChecked(), options.doXGeneration,
refoptions.doXGeneration);
        changes += changeOption(mJSONFile.getText(), options.xfilename,
refoptions.xfilename);
```

```
        return changes;
    }

    CheckBox mGenDoc;
    Text mDocDir;
    Text mDocFile;
    Text mModulesDDoc;
    CheckBox mGenHdr;
    Text mHdrDir;
    Text mHdrFile;
    CheckBox mGenJSON;
    Text mJSONFile;
}

class DmdOutputPropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return "Compiler"; }
    override string GetPageName() { return "Output"; }

    override void CreateControls()
    {
        mUnindentCheckBox = kLabelWidth;
        AddControl("", mMultiObj = new CheckBox(mCanvas, "Multiple Object Files"));
        AddControl("", mPreservePaths = new CheckBox(mCanvas, "Keep Path From Source
File"));
        AddControl("", mMsCoff32 = new CheckBox(mCanvas, "Use MS-COFF object file format
for Win32 (DMD 2.067+)"));
    }

    override void SetControls(ProjectOptions options)
    {
        mMultiObj.setChecked(options.multiobj);
        mPreservePaths.setChecked(options.preservePaths);
        mMsCoff32.setChecked(options.mscoff);
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
```

```
        changes += changeOption(mMultiObj.isChecked(), options.multiobj, refoptions.multiobj);
        changes += changeOption(mPreservePaths.isChecked(), options.preservePaths,
refoptions.preservePaths);
        changes += changeOption(mMsCoff32.isChecked(), options.mscoff, refoptions.mscoff);
        return changes;
    }

    CheckBox mMultiObj;
    CheckBox mPreservePaths;
    CheckBox mMsCoff32;
}

class DmdLinkerPropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return "Linker"; }
    override string GetPageName() { return "General"; }

    this()
    {
        kNeededLines = 11;
    }

    override void UpdateDirty(bool bDirty)
    {
        super.UpdateDirty(bDirty);
        EnableControls();
    }

    enum ID_OBJECTFILES = 1050;
    enum ID_LIBRARYFILES = 1051;
    enum ID_LIBRARYPATHS = 1052;
    enum ID_DEFFILE = 1053;
    enum ID_RESFILE = 1054;

    extern(D) override void OnCommand(Widget w, int cmd)
    {
        switch(cmd)
        {
            case ID_OBJECTFILES:
```

```
            addBrowsePath(mObjFiles, false, GetProjectDir(), " ", "Add object file", "Object
files\0*.obj\0All Files\0*.*\0");
            break;
        case ID_LIBRARYFILES:
            addBrowsePath(mLibFiles, false, GetProjectDir(), " ", "Add library file", "Library
files\0*.lib\0All Files\0*.*\0");
            break;
        case ID_LIBRARYPATHS:
            addBrowsePath(mLibPaths, true, GetProjectDir(), " ", "Add library path");
            break;

        case ID_DEFFILE:
            if(auto file = browseFile(mCanvas.hwnd, "Select definition file", "Definition
files\0*.def\0All Files\0*.*\0", GetProjectDir()))
                mDefFile.setText(makeRelative(file, GetProjectDir()));
            break;
        case ID_RESFILE:
            if(auto file = browseFile(mCanvas.hwnd, "Select resource file", "Resource
files\0*.res\0All Files\0*.*\0", GetProjectDir()))
                mResFile.setText(makeRelative(file, GetProjectDir()));
            break;
        default:
            break;
      }
      super.OnCommand(w, cmd);
   }

   override void CreateControls()
   {
      AddControl("Output File", mExeFile = new Text(mCanvas));
      auto btn = new Button(mCanvas, "+", ID_OBJECTFILES);
      AddControl("Object Files", mObjFiles = new Text(mCanvas), btn);
      btn = new Button(mCanvas, "+", ID_LIBRARYFILES);
      AddControl("Library Files", mLibFiles = new Text(mCanvas), btn);
      btn = new Button(mCanvas, "+", ID_LIBRARYPATHS);
      AddControl("Library Search Path", mLibPaths = new Text(mCanvas), btn);
      //AddControl("Library search paths only work if you have modified sc.ini to include
DMD_LIB!", null);
      btn = new Button(mCanvas, "...", ID_DEFFILE);
```

```
        AddControl("Definition File", mDefFile = new Text(mCanvas), btn);
        btn = new Button(mCanvas, "...", ID_RESFILE);
        AddControl("Resource File",   mResFile = new Text(mCanvas), btn);
        AddControl("Generate Map File", mGenMap = new ComboBox(mCanvas,
            [ "Minimum", "Symbols By Address", "Standard", "Full", "With cross references" ],
false));
        AddControl("", mImplib = new CheckBox(mCanvas, "Create import library"));
        AddControl("", mPrivatePhobos = new CheckBox(mCanvas, "Build and use local version
of phobos with same compiler options"));
        AddControl("", mUseStdLibPath = new CheckBox(mCanvas, "Use global and standard
library search paths"));
        AddControl("C Runtime", mCRuntime = new ComboBox(mCanvas, [ "None", "Static
Release (LIBCMT)", "Static Debug (LIBCMTD)", "Dynamic Release (MSCVRT)", "Dynamic
Debug (MSCVRTD)" ], false));
    }

    void EnableControls()
    {
        if(ProjectOptions options = GetProjectOptions())
            mCRuntime.setEnabled(options.isX86_64 || options.mscoff);
    }

    override void SetControls(ProjectOptions options)
    {
        mExeFile.setText(options.exefile);
        mObjFiles.setText(options.objfiles);
        mLibFiles.setText(options.libfiles);
        mLibPaths.setText(options.libpaths);
        mDefFile.setText(options.deffile);
        mResFile.setText(options.resfile);
        mGenMap.setSelection(options.mapverbosity);
        mImplib.setChecked(options.createImplib);
        mUseStdLibPath.setChecked(options.useStdLibPath);
        mPrivatePhobos.setChecked(options.privatePhobos);
        mCRuntime.setSelection(options.cRuntime);

        EnableControls();
    }
```

```
    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
        changes += changeOption(mExeFile.getText(), options.exefile, refoptions.exefile);
        changes += changeOption(mObjFiles.getText(), options.objfiles, refoptions.objfiles);
        changes += changeOption(mLibFiles.getText(), options.libfiles, refoptions.libfiles);
        changes += changeOption(mLibPaths.getText(), options.libpaths, refoptions.libpaths);
        changes += changeOption(mDefFile.getText(), options.deffile, refoptions.deffile);
        changes += changeOption(mResFile.getText(), options.resfile, refoptions.resfile);
        changes += changeOption(cast(uint) mGenMap.getSelection(), options.mapverbosity,
refoptions.mapverbosity);
        changes += changeOption(mImplib.isChecked(), options.createImplib,
refoptions.createImplib);
        changes += changeOption(mUseStdLibPath.isChecked(), options.useStdLibPath,
refoptions.useStdLibPath);
        changes += changeOption(mPrivatePhobos.isChecked(), options.privatePhobos,
refoptions.privatePhobos);
        changes += changeOption(cast(uint) mCRuntime.getSelection(), options.cRuntime,
refoptions.cRuntime);
        return changes;
    }

    Text mExeFile;
    Text mObjFiles;
    Text mLibFiles;
    Text mLibPaths;
    Text mDefFile;
    Text mResFile;
    ComboBox mGenMap;
    CheckBox mImplib;
    CheckBox mUseStdLibPath;
    CheckBox mPrivatePhobos;
    ComboBox mCRuntime;
}

class DmdEventsPropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return ""; }
    override string GetPageName() { return "Build Events"; }
```

```
    override void CreateControls()
    {
        mLinesPerMultiLine = 5;
        AddControl("Pre-Build Command", mPreCmd = new MultiLineText(mCanvas), 500);
        AddControl("Post-Build Command", mPostCmd = new MultiLineText(mCanvas), 500);

        Label lab = new Label(mCanvas, "Use \"if errorlevel 1 goto reportError\" to cancel on
error");
        lab.setRect(0, mLineY, getWidgetWidth(mCanvas, kPageWidth), kLineHeight);
        addResizableWidget(lab, kAttachBottom);
    }

    override void SetControls(ProjectOptions options)
    {
        mPreCmd.setText(options.preBuildCommand);
        mPostCmd.setText(options.postBuildCommand);
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
        changes += changeOption(mPreCmd.getText(), options.preBuildCommand,
refoptions.preBuildCommand);
        changes += changeOption(mPostCmd.getText(), options.postBuildCommand,
refoptions.postBuildCommand);
        return changes;
    }

    MultiLineText mPreCmd;
    MultiLineText mPostCmd;
}

class DmdCmdLinePropertyPage : ProjectPropertyPage
{
    override string GetCategoryName() { return ""; }
    override string GetPageName() { return "Command line"; }

    override void CreateControls()
```

```
    {
        mLinesPerMultiLine = 5;
        AddControl("Command line", mCmdLine = new MultiLineText(mCanvas, "", 0, true), 500);
        AddControl("Additional options", mAddOpt = new MultiLineText(mCanvas), 500);
    }

    override void OnConfigModified()
    {
        if(ProjectOptions options = GetProjectOptions())
            if(mCmdLine && mCmdLine.hwnd)
                mCmdLine.setText(options.buildCommandLine(GetConfig(), true, true,
options.getDependenciesPath()));
    }

    override void SetControls(ProjectOptions options)
    {
        mCmdLine.setText(options.buildCommandLine(GetConfig(), true, true,
options.getDependenciesPath()));
        mAddOpt.setText(options.additionalOptions);
    }

    override int DoApply(ProjectOptions options, ProjectOptions refoptions)
    {
        int changes = 0;
        changes += changeOption(mAddOpt.getText(), options.additionalOptions,
refoptions.additionalOptions);
        return changes;
    }

    MultiLineText mCmdLine;
    MultiLineText mAddOpt;
}

class ConfigNodePropertyPage : ProjectPropertyPage
{
    abstract void SetControls(CFileNode node);
    abstract int  DoApply(CFileNode node, CFileNode refnode, Config cfg);

    override void SetControls(ProjectOptions options)
```

```
{
    mNodes = GetSelectedNodes();
    if(auto node = GetNode())
        SetControls(node);
}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)
{
    return 0;
}

CHierNode[] GetSelectedNodes()
{
    if(auto cfg = GetConfig()) // any config works
    {
        auto prj = cfg.GetProject();
        CHierNode[] nodes;
        prj.GetSelectedNodes(nodes);
        return nodes;
    }
    return null;
}

CFileNode GetNode()
{
    for(size_t i = 0; i < mNodes.length; i++)
        if(auto node = cast(CFileNode)mNodes[i])
            return node;
    return null;
}

override int IsPageDirty()
{
    mixin(LogCallMix);
    if(mWindow)
        if(CFileNode node = GetNode())
        {
            Config cfg = GetConfig();
            scope CFileNode n = newCom!CFileNode("");
```

```
            return DoApply(n, node, cfg) > 0 ? S_OK : S_FALSE;
        }
      return S_FALSE;
   }

   override int Apply()
   {
      mixin(LogCallMix);

      if(CFileNode rnode = GetNode())
      {
         auto refnode = rnode.cloneDeep();
         for(int i = 0; i < mObjects.length; i++)
         {
            auto config = ComPtr!(Config)(mObjects[i]);
            if(config)
            {
               for(size_t n = 0; n < mNodes.length; n++)
                  if(auto node = cast(CFileNode)mNodes[n])
                  {
                     DoApply(node, refnode, config);
                     if(CProjectNode pn = cast(CProjectNode) node.GetRootNode())
                        pn.SetProjectFileDirty(true);
                  }
            }
            return S_OK;
         }
      }
      return returnError(E_FAIL);
   }

   CHierNode[] mNodes;
}

class FilePropertyPage : ConfigNodePropertyPage
{
   override string GetCategoryName() { return ""; }
   override string GetPageName() { return "File"; }
```

```
    override void CreateControls()
    {
        mLinesPerMultiLine = 3;
        AddControl("", mPerConfig = new CheckBox(mCanvas, "per Configuration Options (apply and reopen dialog to update)"));
        AddControl("Build Tool", mTool = new ComboBox(mCanvas, [ "Auto", "DMD", kToolCpp, kToolResourceCompiler, "Custom", "None" ], false));
        AddControl("Additional Options", mAddOpt = new Text(mCanvas));
        AddControl("Build Command", mCustomCmd = new MultiLineText(mCanvas), 1000);
        AddControl("Other Dependencies", mDependencies = new Text(mCanvas));
        AddControl("Output File", mOutFile = new Text(mCanvas));
        AddControl("", mLinkOut = new CheckBox(mCanvas, "Add output to link"));
        AddControl("", mUptodateWithSameTime = new CheckBox(mCanvas, "Assume output up to date with same time as input"));
    }

    override void UpdateDirty(bool bDirty)
    {
        super.UpdateDirty(bDirty);

        enableControls(mTool.getText());
    }

    void enableControls(string tool)
    {
        bool perConfigChanged = mInitPerConfig != mPerConfig.isChecked();
        bool isCustom = (tool == "Custom");
        bool isRc = (tool == kToolResourceCompiler);
        bool isCpp = (tool == kToolCpp);
        mTool.setEnabled(!perConfigChanged);
        mCustomCmd.setEnabled(!perConfigChanged && isCustom);
        mAddOpt.setEnabled(!perConfigChanged && (isRc || isCpp));
        mDependencies.setEnabled(!perConfigChanged && (isCustom || isRc));
        mOutFile.setEnabled(!perConfigChanged && isCustom);
        mLinkOut.setEnabled(!perConfigChanged && isCustom);
        mUptodateWithSameTime.setEnabled(!perConfigChanged && isCustom);
    }

    string GetCfgName()
```

```
    {
        return GetConfig().getCfgName();
    }

    override void SetControls(CFileNode node)
    {
        string cfgname = GetCfgName();
        string tool = node.GetTool(cfgname);
        if(tool.length == 0)
            mTool.setSelection(0);
        else
            mTool.setSelection(mTool.findString(tool));

        mInitPerConfig = node.GetPerConfigOptions();
        mPerConfig.setChecked(mInitPerConfig);
        mCustomCmd.setText(node.GetCustomCmd(cfgname));
        mAddOpt.setText(node.GetAdditionalOptions(cfgname));
        mDependencies.setText(node.GetDependencies(cfgname));
        mOutFile.setText(node.GetOutFile(cfgname));
        mLinkOut.setChecked(node.GetLinkOutput(cfgname));
        mUptodateWithSameTime.setChecked(node.GetUptodateWithSameTime(cfgname));

        enableControls(tool);
    }

    override int DoApply(CFileNode node, CFileNode refnode, Config cfg)
    {
        string cfgname = GetCfgName();
        int changes = 0;
        string tool = mTool.getText();
        if(tool == "Auto")
            tool = "";
        changes += changeOptionDg!bool(mPerConfig.isChecked(),
&node.SetPerConfigOptions, refnode.GetPerConfigOptions());
        changes += changeOptionDg!string(tool,                  (s) => node.SetTool(cfgname,
s),        refnode.GetTool(cfgname));
        changes += changeOptionDg!string(mCustomCmd.getText(),    (s) =>
node.SetCustomCmd(cfgname, s),    refnode.GetCustomCmd(cfgname));
        changes += changeOptionDg!string(mAddOpt.getText(),       (s) =>
```

```
node.SetAdditionalOptions(cfgname, s), refnode.GetAdditionalOptions(cfgname));
        changes += changeOptionDg!string(mDependencies.getText(), (s) =>
node.SetDependencies(cfgname, s), refnode.GetDependencies(cfgname));
        changes += changeOptionDg!string(mOutFile.getText(),      (s) =>
node.SetOutFile(cfgname, s),      refnode.GetOutFile(cfgname));
        changes += changeOptionDg!bool(mLinkOut.isChecked(),      (b) =>
node.SetLinkOutput(cfgname, b),   refnode.GetLinkOutput(cfgname));
        changes += changeOptionDg!bool(mUptodateWithSameTime.isChecked(),
                                  (b) => node.SetUptodateWithSameTime(cfgname, b),
refnode.GetUptodateWithSameTime(cfgname));
        enableControls(tool);
        return changes;
    }

    bool mInitPerConfig;

    CheckBox mPerConfig;
    ComboBox mTool;
    MultiLineText mCustomCmd;
    Text mAddOpt;
    Text mDependencies;
    Text mOutFile;
    CheckBox mLinkOut;
    CheckBox mUptodateWithSameTime;
}

////////////////////////////////////////////////////////////////////////////
class DirPropertyPage : GlobalPropertyPage
{
    enum ID_BROWSEINSTALLDIR = 1000;
    enum ID_IMPORTDIR = 1001;
    enum ID_EXEPATH32 = 1002;
    enum ID_EXEPATH64 = 1003;
    enum ID_EXEPATH32COFF = 1004;
    enum ID_LIBPATH32 = 1005;
    enum ID_LIBPATH64 = 1006;
    enum ID_LIBPATH32COFF = 1007;
    enum ID_LINKER64 = 1008;
    enum ID_LINKER32COFF = 1009;
```

```
this(GlobalOptions options)
{
    super(options);
    kNeededLines = 13;
}

void addBrowseDir(MultiLineText ctrl, string title)
{
    addBrowsePath(ctrl, true, null, "\n", title);
}

extern(D) override void OnCommand(Widget w, int cmd)
{
    switch(cmd)
    {
        case ID_BROWSEINSTALLDIR:
            if(auto dir = browseDirectory(mCanvas.hwnd, "Select installation directory"))
                mDmdPath.setText(dir);
            break;
        case ID_IMPORTDIR:
            addBrowseDir(mImpPath, "Add import directory");
            break;

        case ID_EXEPATH32:
            addBrowseDir(mExePath, "Add executable directory");
            break;
        case ID_EXEPATH64:
            addBrowseDir(mExePath64, "Add executable directory");
            break;
        case ID_EXEPATH32COFF:
            addBrowseDir(mExePath32coff, "Add executable directory");
            break;

        case ID_LIBPATH32:
            addBrowseDir(mLibPath, "Add library directory");
            break;
        case ID_LIBPATH64:
            addBrowseDir(mLibPath64, "Add library directory");
```

```
            break;
        case ID_LIBPATH32COFF:
            addBrowseDir(mLibPath32coff, "Add library directory");
            break;

        case ID_LINKER64:
            if(auto file = browseFile(mCanvas.hwnd, "Select linker executable",
"Executables\0*.exe\0All Files\0*.*\0"))
                mLinkerExecutable64.setText(file);
            break;
        case ID_LINKER32COFF:
            if(auto file = browseFile(mCanvas.hwnd, "Select linker executable",
"Executables\0*.exe\0All Files\0*.*\0"))
                mLinkerExecutable32coff.setText(file);
            break;
        default:
            break;
    }
    super.OnCommand(w, cmd);
  }

  void dirCreateControls(string name, string overrideIni)
  {
    auto btn = new Button(mCanvas, "...", ID_BROWSEINSTALLDIR);
    AddControl(name ~ " install path", mDmdPath = new Text(mCanvas), btn);
    mLinesPerMultiLine = 2;
    btn = new Button(mCanvas, "+", ID_IMPORTDIR);
    AddControl("Import paths",    mImpPath = new MultiLineText(mCanvas), btn, 300);

    mLinesPerMultiLine = 10;
    string[] archs = ["Win32", "x64"];
    if(overrideIni.length)
        archs ~= "Win32-COFF";
    AddControl("", mTabArch = new TabControl(mCanvas, archs), 700);

    auto page32 = mTabArch.pages[0];
    if(auto w = cast(Window)page32)
        w.commandDelegate = mCanvas.commandDelegate;
```

```
mLineY = 0;
mAttachY = 0;
mLinesPerMultiLine = 3;
btn = new Button(page32, "+", ID_EXEPATH32);
AddControl("Executable paths", mExePath = new MultiLineText(page32), btn, 500);
mLinesPerMultiLine = 2;
btn = new Button(page32, "+", ID_LIBPATH32);
AddControl("Library paths",    mLibPath = new MultiLineText(page32), btn, 500);
AddControl("Disassemble Command", mDisasmCommand = new Text(page32));

auto page64 = mTabArch.pages[1];
if(auto w = cast(Window)page64)
    w.commandDelegate = mCanvas.commandDelegate;

mLineY = 0;
mAttachY = 0;
mLinesPerMultiLine = 3;
btn = new Button(page64, "+", ID_EXEPATH64);
AddControl("Executable paths", mExePath64 = new MultiLineText(page64), btn, 500);
mLinesPerMultiLine = 2;
btn = new Button(page64, "+", ID_LIBPATH64);
AddControl("Library paths", mLibPath64 = new MultiLineText(page64), btn, 500);
AddControl("Disassemble Command", mDisasmCommand64 = new Text(page64));

if(overrideIni.length)
{
    AddControl("", mOverrideIni64 = new CheckBox(page64, overrideIni));
    btn = new Button(page64, "...", ID_LINKER64);
    AddControl("Linker", mLinkerExecutable64 = new Text(page64), btn);
    AddControl("Additional options", mLinkerOptions64 = new Text(page64));

    auto page32coff = mTabArch.pages[2];
    if(auto w = cast(Window)page32coff)
        w.commandDelegate = mCanvas.commandDelegate;

    mLineY = 0;
    mAttachY = 0;
    mLinesPerMultiLine = 3;
    btn = new Button(page32coff, "+", ID_EXEPATH32COFF);
```

```
        AddControl("Executable paths", mExePath32coff = new MultiLineText(page32coff), btn,
500);
        mLinesPerMultiLine = 2;
        btn = new Button(page32coff, "+", ID_LIBPATH32COFF);
        AddControl("Library paths", mLibPath32coff = new MultiLineText(page32coff), btn,
500);
        AddControl("Disassemble Command", mDisasmCommand32coff = new
Text(page32coff));

        AddControl("", mOverrideIni32coff = new CheckBox(page32coff, overrideIni));
        btn = new Button(page32coff, "...", ID_LINKER32COFF);
        AddControl("Linker", mLinkerExecutable32coff = new Text(page32coff), btn);
        AddControl("Additional options", mLinkerOptions32coff = new Text(page32coff));
    }
  }

  override void UpdateDirty(bool bDirty)
  {
    super.UpdateDirty(bDirty);

    enableControls();
  }

  void enableControls()
  {
    if(mOverrideIni64)
    {
      mLinkerExecutable64.setEnabled(mOverrideIni64.isChecked());
      mLinkerOptions64.setEnabled(mOverrideIni64.isChecked());
    }
    if(mOverrideIni32coff)
    {
      mLinkerExecutable32coff.setEnabled(mOverrideIni32coff.isChecked());
      mLinkerOptions32coff.setEnabled(mOverrideIni32coff.isChecked());
    }
  }

  abstract CompilerDirectories* getCompilerOptions(GlobalOptions opts);
```

```
    override void SetControls(GlobalOptions opts)
    {
        CompilerDirectories* opt = getCompilerOptions(opts);

        mDmdPath.setText(opt.InstallDir);
        mExePath.setText(opt.ExeSearchPath);
        mImpPath.setText(opt.ImpSearchPath);
        mLibPath.setText(opt.LibSearchPath);
        mDisasmCommand.setText(opt.DisasmCommand);
        mExePath64.setText(opt.ExeSearchPath64);
        mLibPath64.setText(opt.LibSearchPath64);
        mDisasmCommand64.setText(opt.DisasmCommand64);
        if(mOverrideIni64)
        {
            mOverrideIni64.setChecked(opt.overrideIni64);
            mLinkerExecutable64.setText(opt.overrideLinker64);
            mLinkerOptions64.setText(opt.overrideOptions64);
        }
        if(mOverrideIni32coff)
        {
            mExePath32coff.setText(opt.ExeSearchPath32coff);
            mLibPath32coff.setText(opt.LibSearchPath32coff);
            mOverrideIni32coff.setChecked(opt.overrideIni32coff);
            mLinkerExecutable32coff.setText(opt.overrideLinker32coff);
            mLinkerOptions32coff.setText(opt.overrideOptions32coff);
            mDisasmCommand32coff.setText(opt.DisasmCommand32coff);
        }

        enableControls();
    }

    override int DoApply(GlobalOptions opts, GlobalOptions refopts)
    {
        CompilerDirectories* opt = getCompilerOptions(opts);
        CompilerDirectories* refopt = getCompilerOptions(refopts);

        int changes = 0;
        changes += changeOption(mDmdPath.getText(),           opt.InstallDir,
refopt.InstallDir);
```

```
        changes += changeOption(mExePath.getText(),           opt.ExeSearchPath,
refopt.ExeSearchPath);
        changes += changeOption(mImpPath.getText(),           opt.ImpSearchPath,
refopt.ImpSearchPath);
        changes += changeOption(mLibPath.getText(),           opt.LibSearchPath,
refopt.LibSearchPath);
        changes += changeOption(mDisasmCommand.getText(),     opt.DisasmCommand,
refopt.DisasmCommand);
        changes += changeOption(mExePath64.getText(),         opt.ExeSearchPath64,
refopt.ExeSearchPath64);
        changes += changeOption(mLibPath64.getText(),         opt.LibSearchPath64,
refopt.LibSearchPath64);
        changes += changeOption(mDisasmCommand64.getText(),   opt.DisasmCommand64,
refopt.DisasmCommand64);
        if(mOverrideIni64)
        {
            changes += changeOption(mOverrideIni64.isChecked(),   opt.overrideIni64,
refopt.overrideIni64);
            changes += changeOption(mLinkerExecutable64.getText(), opt.overrideLinker64,
refopt.overrideLinker64);
            changes += changeOption(mLinkerOptions64.getText(),   opt.overrideOptions64,
refopt.overrideOptions64);
        }
        if(mOverrideIni32coff)
        {
            changes += changeOption(mExePath32coff.getText(),
opt.ExeSearchPath32coff,   refopt.ExeSearchPath32coff);
            changes += changeOption(mLibPath32coff.getText(),         opt.LibSearchPath32coff,
refopt.LibSearchPath32coff);
            changes += changeOption(mOverrideIni32coff.isChecked(),   opt.overrideIni32coff,
refopt.overrideIni32coff);
            changes += changeOption(mLinkerExecutable32coff.getText(),
opt.overrideLinker32coff,  refopt.overrideLinker32coff);
            changes += changeOption(mLinkerOptions32coff.getText(),
opt.overrideOptions32coff, refopt.overrideOptions32coff);
            changes += changeOption(mDisasmCommand32coff.getText(),
opt.DisasmCommand32coff,   refopt.DisasmCommand32coff);
        }
        return changes;
```

```
    }

    TabControl mTabArch;
    Text mDmdPath;
    MultiLineText mExePath;
    MultiLineText mImpPath;
    MultiLineText mLibPath;
    Text mDisasmCommand;

    MultiLineText mExePath64;
    MultiLineText mLibPath64;
    CheckBox mOverrideIni64;
    Text mLinkerExecutable64;
    Text mLinkerOptions64;
    Text mDisasmCommand64;

    MultiLineText mExePath32coff;
    MultiLineText mLibPath32coff;
    CheckBox mOverrideIni32coff;
    Text mLinkerExecutable32coff;
    Text mLinkerOptions32coff;
    Text mDisasmCommand32coff;
}

////////////////////////////////////////////////////////////////////////////
class DmdDirPropertyPage : DirPropertyPage
{
    override string GetCategoryName() { return "D Options"; }
    override string GetPageName() { return "DMD Directories"; }

    this(GlobalOptions options)
    {
        super(options);
    }

    override void CreateControls()
    {
        dirCreateControls("DMD", "override linker settings from dmd configuration in sc.ini.");
    }
```

```
    override CompilerDirectories* getCompilerOptions(GlobalOptions opts)
    {
        return &opts.DMD;
    }
}

////////////////////////////////////////////////////////////////////////////
class GdcDirPropertyPage : DirPropertyPage
{
    override string GetCategoryName() { return "D Options"; }
    override string GetPageName() { return "GDC Directories"; }

    this(GlobalOptions options)
    {
        super(options);
    }

    override void CreateControls()
    {
        dirCreateControls("GDC", "");
    }

    override CompilerDirectories* getCompilerOptions(GlobalOptions opts)
    {
        return &opts.GDC;
    }
}

////////////////////////////////////////////////////////////////////////////
class LdcDirPropertyPage : DirPropertyPage
{
    override string GetCategoryName() { return "D Options"; }
    override string GetPageName() { return "LDC Directories"; }

    this(GlobalOptions options)
    {
        super(options);
    }
```

```
    override void CreateControls()
    {
        dirCreateControls("LDC", "");
    }

    override CompilerDirectories* getCompilerOptions(GlobalOptions opts)
    {
        return &opts.LDC;
    }
}

////////////////////////////////////////////////////////////////////////////
class ToolsProperty2Page : GlobalPropertyPage
{
    override string GetCategoryName() { return "Projects"; }
    override string GetPageName() { return "D Options"; }

    this(GlobalOptions options)
    {
        super(options);
        kNeededLines = 14;
    }

    override void CreateControls()
    {
        AddControl("", mElasticSpace  = new CheckBox(mCanvas, "Use Elastic Space"));
        AddControl("", mSortProjects  = new CheckBox(mCanvas, "Sort project items"));
        AddControl("", mShowUptodate  = new CheckBox(mCanvas, "Show why a target is
rebuilt"));
        AddControl("", mTimeBuilds    = new CheckBox(mCanvas, "Show build time"));
        static if(enableShowMemUsage)
            AddControl("", mShowMemUsage  = new CheckBox(mCanvas, "Show compiler
memory usage"));
        AddControl("", mStopSlnBuild  = new CheckBox(mCanvas, "Stop solution build on
error"));
        AddHorizontalLine();
        AddControl("", mDemangleError = new CheckBox(mCanvas, "Demangle names in link
errors/disassembly"));
```

```
        AddControl("", mOptlinkDeps   = new CheckBox(mCanvas, "Monitor linker
dependencies"));
        mLinesPerMultiLine = 2;
        AddControl("Exclude Files/Folders", mExcludeFileDeps = new MultiLineText(mCanvas));
        AddHorizontalLine();
        //AddControl("Remove project item", mDeleteFiles =
        //          new ComboBox(mCanvas, [ "Do not delete file on disk", "Ask", "Delete file on
disk" ]));
        mLinesPerMultiLine = 2;
        AddControl("JSON paths",        mJSNPath = new MultiLineText(mCanvas));
        AddControl("Resource includes", mIncPath = new Text(mCanvas));
        AddHorizontalLine();
        AddControl("Compile+Run options", mCompileAndRunOpts = new Text(mCanvas));
        AddControl("Compile+Debug options", mCompileAndDbgOpts = new Text(mCanvas));
        AddControl("   Debugger",       mCompileAndDbgEngine = new ComboBox(mCanvas, [
"Visual Studio", "Mago", "Visual Studio (x86 Mixed Mode)" ], false));
    }

    override void SetControls(GlobalOptions opts)
    {
        mTimeBuilds.setChecked(opts.timeBuilds);
        mElasticSpace.setChecked(opts.elasticSpace);
        mSortProjects.setChecked(opts.sortProjects);
        mShowUptodate.setChecked(opts.showUptodateFailure);
        mStopSlnBuild.setChecked(opts.stopSolutionBuild);
        mDemangleError.setChecked(opts.demangleError);
        mOptlinkDeps.setChecked(opts.optlinkDeps);
        mExcludeFileDeps.setText(opts.excludeFileDeps);
        static if(enableShowMemUsage)
            mShowMemUsage.setChecked(opts.showMemUsage);
        //mDeleteFiles.setSelection(opts.deleteFiles + 1);
        mIncPath.setText(opts.IncSearchPath);
        mJSNPath.setText(opts.JSNSearchPath);
        mCompileAndRunOpts.setText(opts.compileAndRunOpts);
        mCompileAndDbgOpts.setText(opts.compileAndDbgOpts);
        mCompileAndDbgEngine.setSelection(opts.compileAndDbgEngine);
    }

    override int DoApply(GlobalOptions opts, GlobalOptions refopts)
```

```
    {
        int changes = 0;
        changes += changeOption(mTimeBuilds.isChecked(), opts.timeBuilds,
refopts.timeBuilds);
        changes += changeOption(mElasticSpace.isChecked(), opts.elasticSpace,
refopts.elasticSpace);
        changes += changeOption(mSortProjects.isChecked(), opts.sortProjects,
refopts.sortProjects);
        changes += changeOption(mShowUptodate.isChecked(), opts.showUptodateFailure,
refopts.showUptodateFailure);
        changes += changeOption(mStopSlnBuild.isChecked(), opts.stopSolutionBuild,
refopts.stopSolutionBuild);
        changes += changeOption(mDemangleError.isChecked(), opts.demangleError,
refopts.demangleError);
        changes += changeOption(mOptlinkDeps.isChecked(), opts.optlinkDeps,
refopts.optlinkDeps);
        changes += changeOption(mExcludeFileDeps.getText(), opts.excludeFileDeps,
refopts.excludeFileDeps);
        static if(enableShowMemUsage)
            changes += changeOption(mShowMemUsage.isChecked(), opts.showMemUsage,
refopts.showMemUsage);
        //changes += changeOption(cast(byte) (mDeleteFiles.getSelection() - 1), opts.deleteFiles,
refopts.deleteFiles);
        changes += changeOption(mIncPath.getText(), opts.IncSearchPath,
refopts.IncSearchPath);
        changes += changeOption(mJSNPath.getText(), opts.JSNSearchPath,
refopts.JSNSearchPath);
        changes += changeOption(mCompileAndRunOpts.getText(), opts.compileAndRunOpts,
refopts.compileAndRunOpts);
        changes += changeOption(mCompileAndDbgOpts.getText(), opts.compileAndDbgOpts,
refopts.compileAndDbgOpts);
        changes += changeOption(mCompileAndDbgEngine.getSelection(),
opts.compileAndDbgEngine, refopts.compileAndDbgEngine);
        return changes;
    }

    CheckBox mTimeBuilds;
    CheckBox mElasticSpace;
    CheckBox mSortProjects;
```

```
    CheckBox mShowUptodate;
    CheckBox mStopSlnBuild;
    CheckBox mDemangleError;
    CheckBox mOptlinkDeps;
    MultiLineText mExcludeFileDeps;
    static if(enableShowMemUsage)
        CheckBox mShowMemUsage;
    //ComboBox mDeleteFiles;
    Text mIncPath;
    Text mCompileAndRunOpts;
    Text mCompileAndDbgOpts;
    ComboBox mCompileAndDbgEngine;
    MultiLineText mJSNPath;
}

////////////////////////////////////////////////////////////////////////////
class ColorizerPropertyPage : GlobalPropertyPage
{
    override string GetCategoryName() { return "Language"; }
    override string GetPageName() { return "Colorizer"; }

    this(GlobalOptions options)
    {
        super(options);
        kNeededLines = 11;
    }

    override void CreateControls()
    {
        AddControl("", mColorizeVersions = new CheckBox(mCanvas, "Colorize version and
debug statements"));
        AddControl("Colored types", mUserTypes = new MultiLineText(mCanvas), 1000);
        AddHorizontalLine();
        AddControl("", mColorizeCoverage = new CheckBox(mCanvas, "Colorize coverage from
.LST file"));
        AddControl("", mShowCoverageMargin = new CheckBox(mCanvas, "Show coverage
margin"));
        AddHorizontalLine();
        AddControl("", mAutoOutlining = new CheckBox(mCanvas, "Add outlining regions when
```

```
opening D files"));
        AddControl("", mParseSource = new CheckBox(mCanvas, "Parse source for syntax
errors"));
        AddControl("", mPasteIndent = new CheckBox(mCanvas, "Reindent new lines after
paste"));
    }

    override void SetControls(GlobalOptions opts)
    {
        mColorizeVersions.setChecked(opts.ColorizeVersions);
        mColorizeCoverage.setChecked(opts.ColorizeCoverage);
        mShowCoverageMargin.setChecked(opts.showCoverageMargin);
        mAutoOutlining.setChecked(opts.autoOutlining);
        mParseSource.setChecked(opts.parseSource);
        mPasteIndent.setChecked(opts.pasteIndent);
        mUserTypes.setText(opts.UserTypesSpec);

        //mSemantics.setEnabled(false);
    }

    override int DoApply(GlobalOptions opts, GlobalOptions refopts)
    {
        int changes = 0;
        changes += changeOption(mColorizeVersions.isChecked(), opts.ColorizeVersions,
refopts.ColorizeVersions);
        changes += changeOption(mColorizeCoverage.isChecked(), opts.ColorizeCoverage,
refopts.ColorizeCoverage);
        changes += changeOption(mShowCoverageMargin.isChecked(),
opts.showCoverageMargin, refopts.showCoverageMargin);
        changes += changeOption(mAutoOutlining.isChecked(), opts.autoOutlining,
refopts.autoOutlining);
        changes += changeOption(mParseSource.isChecked(), opts.parseSource,
refopts.parseSource);
        changes += changeOption(mPasteIndent.isChecked(), opts.pasteIndent,
refopts.pasteIndent);
        changes += changeOption(mUserTypes.getText(), opts.UserTypesSpec,
refopts.UserTypesSpec);
        return changes;
    }
```

```
    CheckBox mColorizeVersions;
    CheckBox mColorizeCoverage;
    CheckBox mShowCoverageMargin;
    CheckBox mAutoOutlining;
    CheckBox mParseSource;
    CheckBox mPasteIndent;
    MultiLineText mUserTypes;
}

////////////////////////////////////////////////////////////////////////////
class IntellisensePropertyPage : GlobalPropertyPage
{
    override string GetCategoryName() { return "Language"; }
    override string GetPageName() { return "Intellisense"; }

    this(GlobalOptions options)
    {
        super(options);
    }

    override void CreateControls()
    {
        AddControl("", mExpandSemantics = new CheckBox(mCanvas, "Expansions from
semantic analysis"));
        AddControl("", mExpandFromBuffer = new CheckBox(mCanvas, "Expansions from text
buffer"));
        AddControl("", mExpandFromJSON = new CheckBox(mCanvas, "Expansions from JSON
browse information"));
        AddControl("Show expansion when", mExpandTrigger = new ComboBox(mCanvas, [
"pressing Ctrl+Space", "writing '.'", "writing an identifier" ], false));
        AddControl("", mShowTypeInTooltip = new CheckBox(mCanvas, "Show type of
expressions in tool tip"));
        AddControl("", mSemanticGotoDef = new CheckBox(mCanvas, "Use semantic analysis
for \"Goto Definition\" (before trying JSON info)"));
        version(DParserOption) AddControl("", mUseDParser = new CheckBox(mCanvas, "Use
Alexander Bothe's D parsing engine for semantic analysis"));
        AddControl("", mMixinAnalysis = new CheckBox(mCanvas, "Enable mixin analysis"));
        AddControl("", mUFCSExpansions = new CheckBox(mCanvas, "Enable UFCS
```

```
expansions"));
        AddControl("Sort expansions", mSortExpMode = new ComboBox(mCanvas, [
"alphabetically", "by type", "by declaration and scope" ], false));
        AddControl("", mExactExpMatch = new CheckBox(mCanvas, "Expansions match exact
start instead of case insensitive sub string"));
    }

    override void UpdateDirty(bool bDirty)
    {
        super.UpdateDirty(bDirty);
        EnableControls();
    }

    void EnableControls()
    {
        version(DParserOption) bool useDParser = mUseDParser.isChecked();
        else                   bool useDParser = true;
        mMixinAnalysis.setEnabled(useDParser);
        mUFCSExpansions.setEnabled(useDParser);
        mSortExpMode.setEnabled(useDParser);
    }

    override void SetControls(GlobalOptions opts)
    {
        mExpandSemantics.setChecked(opts.expandFromSemantics);
        mExpandFromBuffer.setChecked(opts.expandFromBuffer);
        mExpandFromJSON.setChecked(opts.expandFromJSON);
        mExpandTrigger.setSelection(opts.expandTrigger);
        mShowTypeInTooltip.setChecked(opts.showTypeInTooltip);
        mSemanticGotoDef.setChecked(opts.semanticGotoDef);
        version(DParserOption) mUseDParser.setChecked(opts.useDParser);
        mMixinAnalysis.setChecked(opts.mixinAnalysis);
        mUFCSExpansions.setChecked(opts.UFCSExpansions);
        mSortExpMode.setSelection(opts.sortExpMode);
        mExactExpMatch.setChecked(opts.exactExpMatch);

        //mExpandSemantics.setEnabled(false);
    }
```

```
    override int DoApply(GlobalOptions opts, GlobalOptions refopts)
    {
        int changes = 0;
        changes += changeOption(mExpandSemantics.isChecked(),
opts.expandFromSemantics, refopts.expandFromSemantics);
        changes += changeOption(mExpandFromBuffer.isChecked(), opts.expandFromBuffer,
refopts.expandFromBuffer);
        changes += changeOption(mExpandFromJSON.isChecked(), opts.expandFromJSON,
refopts.expandFromJSON);
        changes += changeOption(cast(byte) mExpandTrigger.getSelection(),
opts.expandTrigger, refopts.expandTrigger);
        changes += changeOption(mShowTypeInTooltip.isChecked(), opts.showTypeInTooltip,
refopts.showTypeInTooltip);
        changes += changeOption(mSemanticGotoDef.isChecked(), opts.semanticGotoDef,
refopts.semanticGotoDef);
        version(DParserOption) changes += changeOption(mUseDParser.isChecked(),
opts.useDParser, refopts.useDParser);
        changes += changeOption(mMixinAnalysis.isChecked(), opts.mixinAnalysis,
refopts.mixinAnalysis);
        changes += changeOption(mUFCSExpansions.isChecked(), opts.UFCSExpansions,
refopts.UFCSExpansions);
        changes += changeOption(cast(ubyte) mSortExpMode.getSelection(), opts.sortExpMode,
refopts.sortExpMode);
        changes += changeOption(mExactExpMatch.isChecked(), opts.exactExpMatch,
refopts.exactExpMatch);
        return changes;
    }

    CheckBox mExpandSemantics;
    CheckBox mExpandFromBuffer;
    CheckBox mExpandFromJSON;
    ComboBox mExpandTrigger;
    CheckBox mShowTypeInTooltip;
    CheckBox mSemanticGotoDef;
    version(DParserOption) CheckBox mUseDParser;
    CheckBox mUFCSExpansions;
    ComboBox mSortExpMode;
    CheckBox mExactExpMatch;
    CheckBox mMixinAnalysis;
```

```
}

////////////////////////////////////////////////////////////////////////////
struct MagoOptions
{
   bool hideInternalNames;
   bool showStaticsInAggr;

   void saveToRegistry()
   {
      scope RegKey keyMago = new RegKey(HKEY_CURRENT_USER,
"SOFTWARE\\MagoDebugger", true);
      keyMago.Set("hideInternalNames", hideInternalNames);
      keyMago.Set("showStaticsInAggr", showStaticsInAggr);
   }

   void loadFromRegistry()
   {
      scope RegKey keyMago = new RegKey(HKEY_CURRENT_USER,
"SOFTWARE\\MagoDebugger", false);
      hideInternalNames = (keyMago.GetDWORD("hideInternalNames", 0) != 0);
      showStaticsInAggr = (keyMago.GetDWORD("showStaticsInAggr", 0) != 0);
   }
}

class MagoPropertyPage : ResizablePropertyPage
{
   override string GetCategoryName() { return "Debugging"; }
   override string GetPageName() { return "Mago"; }

   MagoOptions mOptions;

   override void CreateControls()
   {
      AddLabel("Changes to these settings only apply to new debugging sessions");
      AddControl("", mHideInternalNames = new CheckBox(mCanvas, "Hide compiler
generated symbols"));
      AddControl("", mShowStaticsInAggr = new CheckBox(mCanvas, "Show static fields in
structs and classes"));
```

```
}

override void UpdateDirty(bool bDirty)
{
   super.UpdateDirty(bDirty);
   EnableControls();
}

override void UpdateControls()
{
   SetControls();
   EnableControls();
}

/*override*/ int IsPageDirty()
{
   mixin(LogCallMix);
   if(mWindow)
   {
      MagoOptions opt;
      return DoApply(opt, mOptions) > 0 ? S_OK : S_FALSE;
   }
   return S_FALSE;
}

/*override*/ int Apply()
{
   mixin(LogCallMix);

   DoApply(mOptions, mOptions);
   mOptions.saveToRegistry();
   return S_OK;
}

void EnableControls()
{
}

void SetControls()
```

```
    {
        mOptions.loadFromRegistry();
        mHideInternalNames.setChecked(mOptions.hideInternalNames);
        mShowStaticsInAggr.setChecked(mOptions.showStaticsInAggr);
    }

    int DoApply(ref MagoOptions opts, ref MagoOptions refopts)
    {
        int changes = 0;
        changes += changeOption(mHideInternalNames.isChecked(), opts.hideInternalNames,
refopts.hideInternalNames);
        changes += changeOption(mShowStaticsInAggr.isChecked(), opts.showStaticsInAggr,
refopts.showStaticsInAggr);
        return changes;
    }

    CheckBox mHideInternalNames;
    CheckBox mShowStaticsInAggr;
}

////////////////////////////////////////////////////////////////////////////
// more guids in dpackage.d starting up to 980f
const GUID    g_GeneralPropertyPage      = uuid("002a2de9-8bb6-484d-9810-
7e4ad4084715");
const GUID    g_DmdGeneralPropertyPage   = uuid("002a2de9-8bb6-484d-9811-
7e4ad4084715");
const GUID    g_DmdDebugPropertyPage     = uuid("002a2de9-8bb6-484d-9812-
7e4ad4084715");
const GUID    g_DmdCodeGenPropertyPage   = uuid("002a2de9-8bb6-484d-9813-
7e4ad4084715");
const GUID    g_DmdMessagesPropertyPage  = uuid("002a2de9-8bb6-484d-9814-
7e4ad4084715");
const GUID    g_DmdOutputPropertyPage    = uuid("002a2de9-8bb6-484d-9815-
7e4ad4084715");
const GUID    g_DmdLinkerPropertyPage    = uuid("002a2de9-8bb6-484d-9816-
7e4ad4084715");
const GUID    g_DmdEventsPropertyPage    = uuid("002a2de9-8bb6-484d-9817-
7e4ad4084715");
const GUID    g_CommonPropertyPage       = uuid("002a2de9-8bb6-484d-9818-
```

```
7e4ad4084715");
const GUID    g_DebuggingPropertyPage    = uuid("002a2de9-8bb6-484d-9819-
7e4ad4084715");
const GUID    g_FilePropertyPage         = uuid("002a2de9-8bb6-484d-981a-7e4ad4084715");
const GUID    g_DmdDocPropertyPage       = uuid("002a2de9-8bb6-484d-981b-
7e4ad4084715");
const GUID    g_DmdCmdLinePropertyPage   = uuid("002a2de9-8bb6-484d-981c-
7e4ad4084715");

// does not need to be registered, created explicitely by package
const GUID    g_DmdDirPropertyPage       = uuid("002a2de9-8bb6-484d-9820-
7e4ad4084715");
const GUID    g_GdcDirPropertyPage       = uuid("002a2de9-8bb6-484d-9824-
7e4ad4084715");
const GUID    g_LdcDirPropertyPage       = uuid("002a2de9-8bb6-484d-9825-
7e4ad4084715");
const GUID    g_ToolsProperty2Page       = uuid("002a2de9-8bb6-484d-9822-
7e4ad4084715");

// registered under Languages\\Language Services\\D\\EditorToolsOptions\\Colorizer, created
explicitely by package
const GUID    g_ColorizerPropertyPage    = uuid("002a2de9-8bb6-484d-9821-
7e4ad4084715");
const GUID    g_IntellisensePropertyPage = uuid("002a2de9-8bb6-484d-9823-
7e4ad4084715");
const GUID    g_MagoPropertyPage         = uuid("002a2de9-8bb6-484d-9826-
7e4ad4084715");

const GUID*[] guids_propertyPages =
[
    &g_GeneralPropertyPage,
    &g_DmdGeneralPropertyPage,
    &g_DmdDebugPropertyPage,
    &g_DmdCodeGenPropertyPage,
    &g_DmdMessagesPropertyPage,
    &g_DmdOutputPropertyPage,
    &g_DmdLinkerPropertyPage,
    &g_DmdEventsPropertyPage,
    &g_CommonPropertyPage,
```

```
    &g_DebuggingPropertyPage,
    &g_FilePropertyPage,
    &g_DmdDocPropertyPage,
    &g_DmdCmdLinePropertyPage,
];

class PropertyPageFactory : DComObject, IClassFactory
{
    static PropertyPageFactory create(CLSID* rclsid)
    {
        foreach(id; guids_propertyPages)
            if(*id == *rclsid)
                return newCom!PropertyPageFactory(rclsid);
        return null;
    }

    this(CLSID* rclsid)
    {
        mClsid = *rclsid;
    }

    override HRESULT QueryInterface(in IID* riid, void** pvObject)
    {
        if(queryInterface2!(IClassFactory) (this, IID_IClassFactory, riid, pvObject))
            return S_OK;
        return super.QueryInterface(riid, pvObject);
    }

    override HRESULT CreateInstance(IUnknown UnkOuter, in IID* riid, void** pvObject)
    {
        PropertyPage ppp;
        assert(!UnkOuter);

            if(mClsid == g_GeneralPropertyPage)
            ppp = newCom!GeneralPropertyPage();
        else if(mClsid == g_DebuggingPropertyPage)
            ppp = newCom!DebuggingPropertyPage();
        else if(mClsid == g_DmdGeneralPropertyPage)
            ppp = newCom!DmdGeneralPropertyPage();
```

```
        else if(mClsid == g_DmdDebugPropertyPage)
            ppp = newCom!DmdDebugPropertyPage();
        else if(mClsid == g_DmdCodeGenPropertyPage)
            ppp = newCom!DmdCodeGenPropertyPage();
        else if(mClsid == g_DmdMessagesPropertyPage)
            ppp = newCom!DmdMessagesPropertyPage();
        else if(mClsid == g_DmdDocPropertyPage)
            ppp = newCom!DmdDocPropertyPage();
        else if(mClsid == g_DmdOutputPropertyPage)
            ppp = newCom!DmdOutputPropertyPage();
        else if(mClsid == g_DmdLinkerPropertyPage)
            ppp = newCom!DmdLinkerPropertyPage();
        else if(mClsid == g_DmdEventsPropertyPage)
            ppp = newCom!DmdEventsPropertyPage();
        else if(mClsid == g_DmdCmdLinePropertyPage)
            ppp = newCom!DmdCmdLinePropertyPage();
        else if(mClsid == g_CommonPropertyPage)
            ppp = newCom!CommonPropertyPage();
        else if(mClsid == g_FilePropertyPage)
            ppp = newCom!FilePropertyPage();
        else
            return E_INVALIDARG;

        return ppp.QueryInterface(riid, pvObject);
    }

    override HRESULT LockServer(in BOOL fLock)
    {
        return S_OK;
    }

    static int GetProjectPages(CAUUID *pPages, bool addFile)
    {
version(all) {
        pPages.cElems = (addFile ? 12 : 11);
        pPages.pElems = cast(GUID*)CoTaskMemAlloc(pPages.cElems*GUID.sizeof);
        if (!pPages.pElems)
            return E_OUTOFMEMORY;
```

```
        int idx = 0;
        if(addFile)
            pPages.pElems[idx++] = g_FilePropertyPage;
        pPages.pElems[idx++] = g_GeneralPropertyPage;
        pPages.pElems[idx++] = g_DebuggingPropertyPage;
        pPages.pElems[idx++] = g_DmdGeneralPropertyPage;
        pPages.pElems[idx++] = g_DmdDebugPropertyPage;
        pPages.pElems[idx++] = g_DmdCodeGenPropertyPage;
        pPages.pElems[idx++] = g_DmdMessagesPropertyPage;
        pPages.pElems[idx++] = g_DmdDocPropertyPage;
        pPages.pElems[idx++] = g_DmdOutputPropertyPage;
        pPages.pElems[idx++] = g_DmdLinkerPropertyPage;
        pPages.pElems[idx++] = g_DmdCmdLinePropertyPage;
        pPages.pElems[idx++] = g_DmdEventsPropertyPage;
        return S_OK;
    } else {
        return returnError(E_NOTIMPL);
    }
    }

    static int GetCommonPages(CAUUID *pPages)
    {
        pPages.cElems = 1;
        pPages.pElems = cast(GUID*)CoTaskMemAlloc(pPages.cElems*GUID.sizeof);
        if (!pPages.pElems)
            return E_OUTOFMEMORY;

        pPages.pElems[0] = g_CommonPropertyPage;
        return S_OK;
    }

    static int GetFilePages(CAUUID *pPages)
    {
        pPages.cElems = 1;
        pPages.pElems = cast(GUID*)CoTaskMemAlloc(pPages.cElems*GUID.sizeof);
        if (!pPages.pElems)
            return E_OUTOFMEMORY;

        pPages.pElems[0] = g_FilePropertyPage;
```

```
        return S_OK;
    }

private:
    GUID mClsid;
}
```